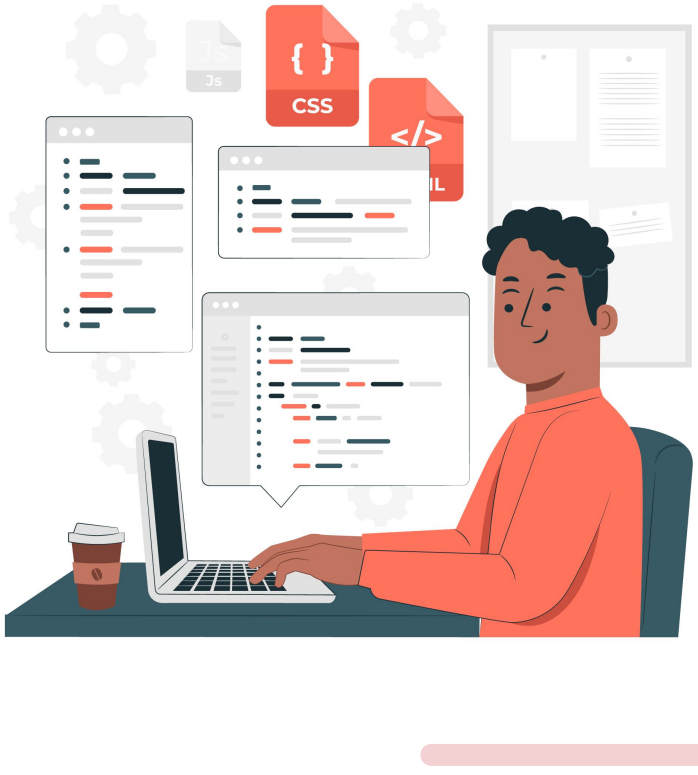
KEMNAKER

Sanbercode

Modern JavaScript: Async & OOP



KEMNAKER

sanbercode

Outline

- Asynchronous
- OOP

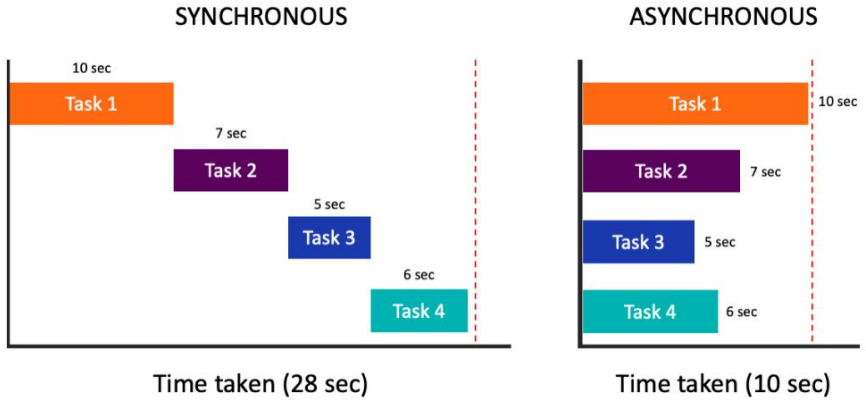
KEMNAKER

Sanbercode

Asynchronous

Asynchronous Javascript

Di dalam dunia pemrograman terdapat dua cara dalam menjalankan program: Synchronous dan Asynchronous. Synchronous artinya program berjalan secara berurutan sedangkan Asynchronous artinya program berjalan bersama-sama. Perhatikanlah ilustrasi dibawah ini :



Apakah Javascript bahasa yang Synchronous saja?

JavaScript adalah bahasa pemrograman yang secara alamiah bersifat synchronous.

Namun, JavaScript juga mendukung operasi asynchronous, yang memungkinkan eksekusi kode untuk melanjutkan tanpa harus menunggu operasi tertentu selesai. Ini terutama terjadi ketika kita berurusan dengan operasi I/O (Input/Output), seperti membaca data dari disk atau mengambil data dari jaringan

Contoh Kode Javascript yang menjadi Asynchronous

Perhatikan contoh program di bawah ini:

```
setTimeout(function() {
  console.log("saya dijalankan belakangan")
}, 3000)

console.log("saya dijalankan pertama")
```

Jika kita jalankan program di atas, maka yang akan tampil terlebih dahulu di console adalah "saya dijalankan pertama" walaupun sintaksnya ditulis belakangan setelah function setTimeout. Function setTimeout di atas merupakan salah satu contoh function asynchronous di Javascript.

Cara mengatasi Asynchronous di Javascript

Cara untuk mengatasi Asynchronous seperti function setTimeout adalah dengan Callback atau dengan Promise.

KEMNAKER

Sanbercode

Callback

Callback Javascript

Callback adalah function yang dipanggil ketika function lain selesai menjalankan programnya.

contoh callback:

```
// Deklarasi function yang memiliki callback sebagai parameter
function periksaDokter(nomerAntri, callback) {
  if(nomerAntri > 50 ) {
      callback(false)
  } else if(nomerAntri < 10) {
      callback(true)
  }
}
```

KEMNAKER

Menjalankan Callback

Setelah deklarasi function yang memiliki callback, kini kita jalankan function tersebut.

```
// Menjalankan function periksaDokter yang sebelumnya sudah dideklarasi
periksaDokter(65, function(check) {
  if(check) {
      console.log("sebentar lagi giliran saya")
  } else {
      console.log("saya jalan-jalan dulu")
  }
})
```

Contoh Lainnya

berikut ini contoh function yang memiliki callback dan terdapat setTimeout didalamnya:

```
function periksaAntrianDokter(nomerAntri, callback) {
  console.log(`sekarang antrian ke-${nomerAntri}`)
  setTimeout(function () {
    if(nomerAntri === 10 ) {
      console.log("saya masuk ruangan dokter")
      callback(0)
    } else {
      console.log("saya masih menunggu")
      callback(nomerAntri+1)
    }
  }, 1000)
}
```

cara menggunakan callback dengan setTimeout di dalamnya adalah seperti ini:

```
periksaAntrianDokter(7, function(nomorAntriBaru){
  return nomorAntriBaru
});
```

jika menggunakan kode diatas maka pada terminal akan muncul

```
sekarang antrian ke-7
saya masih menunggu
```

lalu jika kita ingin membuat kondisi bisa masuk ke ruangan dokter maka kita coba seperti contoh dibawah ini:

```
periksaAntrianDokter(7, function(nomorAntriBaru){
  periksaAntrianDokter(nomorAntriBaru, function(nomorAntriBaru1){
    periksaAntrianDokter(nomorAntriBaru1, function(nomorAntriBaru2){
      periksaAntrianDokter(nomorAntriBaru2, function(nomorAntriBaru3){
        return nomorAntriBaru3
      })
    })
  })
});
```

jika menggunakan kode diatas maka akan sampai ke “saya masuk ruangan dokter”

Waspadai Callback Hell

Jika kita menggunakan contoh pada kode sebelumnya makan akan berhasil tetapi kita perlu memanggil fungsi periksaAntrianDokter sekitar 4 kali berulang, bagaimana jika kita harus melakukan hal tersebut lebih dari itu bahkan ribuan kali.

Memanggil Callback dengan recursive function

maka kita bisa menggunakan recursive function untuk memanggil callback tersebut seperti contoh dibawah ini:

```
function execute(nomorAntri){
  periksaAntrianDokter(nomorAntri, function(nomorAntriBaru){
    if (nomorAntriBaru !== 0){
      execute(nomorAntriBaru)
    }
  })
}

execute(7)
```

KEMNAKER

Sanbercode

Sesi Tanya Jawab

KEMNAKER

Sanbercode

Promise

Promise

Sesuai dengan namanya, Promise berarti janji. Seperti janji yang biasanya memakan waktu dan janji bisa ditepati (resolve) atau diingkari (reject). contoh

```
var isMomHappy = false;

// Promise
var willIGetNewPhone = new Promise(
    function (resolve, reject) {
        if (isMomHappy) {
            var phone = {
                brand: 'Samsung',
                color: 'black'
            };
            resolve(phone); // fulfilled atau janji dipenuhi
        } else {
            var reason = new Error('mom is not happy');
            reject(reason); // reject (ingkar)
        }

    }
);
```

Menjalankan Promise

cara menggunakan Promise pada kode sebelumnya seperti dibawah ini:

```
function askMom() {
  willIGetNewPhone
      .then(function (fulfilled) {
          console.log(fulfilled);
      })
      .catch(function (error) {
          console.log(error.message);
      });
}

askMom()
```

Contoh Promise Lainnya

contoh function yang memiliki return promise:

```
function periksaDataPasien(nomorIdPasien) {
  var dataPasien = [
    {id: 1, nama: "John", jenisKelamin: "Laki-laki"},
    {id: 2, nama: "Michael", jenisKelamin: "Laki-laki"},
    {id: 3, nama: "Sarah", jenisKelamin: "Perempuan"},
    {id: 4, nama: "Frank", jenisKelamin: "Laki-laki"}
  ]
  return new Promise( function (resolve, reject){
    var pasien = dataPasien.find(x=> x.id === nomorIdPasien)
    if (pasien === undefined){
      reject("data pasien tidak ada")
    }else{
      resolve(pasien)
    }
  })
}
```

Cara menggunakan Promise yang memiliki parameter

cara menggunakan promise tersebut adalah seperti dibawah ini:

```
periksaDataPasien(5).then(function(data){
  console.log(data)
}).catch(function(err){
  console.log(err)
})
```

KEMNAKER

Sanbercode

Async/Await

Apa itu Async/Await?

Async/await adalah fitur yang hadir sejak ES2017. Fitur ini mempermudah kita dalam menangani proses asynchronous. untuk penggunaannya misalkan kita memiliki promise seperti dibawah ini:

```
function doAsync() {
  return new Promise( function (resolve, reject){
    var check = true
    if (check){
      resolve("berhasil")
    }else{
      reject("gagal")
    }
  })
}
```

Cara menggunakan Async/Await

cara menggunakan promise tersebut dengan async/await seperti dibawah ini:

```
async function hello(){
    var result = await doAsync()
    console.log(result)
}

hello()
```

Error Handling

ketika menggunakan promise maka pasangan dari then adalah catch yang di mana catch itu adalah error handling dari promise, tapi bagaiamana dengan async/await, async/await menggunakan try dan catch untuk error handlingnya seperti contoh di bawah ini:

```
async function hello(){
  try {
    var result = await doAsync()
    console.log(result)
  } catch(err){
    console.log(err)
  }
}

hello()
```

KEMNAKER

Sanbercode

Sesi Tanya Jawab

KEMNAKER

Sanbercode

OOP di Javascript

Object Oriented Programming (OOP)

Di dalam dunia pemrograman dikenal sebuah konsep dengan nama OOP atau Object Oriented Programming. Secara sederhana, dengan konsep OOP maka segala sesuatu dapat kita anggap sebagai sebuah Object.

KEMNAKER

Class dalam konsep OOP

Class dalam OOP adalah sebuah blueprint atau template untuk menciptakan object-object yang memiliki atribut (data) dan metode (fungsi) yang terkait.

Class adalah cara untuk mendefinisikan struktur atau tipe data yang akan digunakan untuk membuat object dalam suatu program.

Contohnya terdapat Object Class bernama "Car" yang berarti mobil. Suatu (cetakan) mobil biasanya memiliki nama dan dibuat oleh sebuah pabrikan otomotif (factory). Selain itu mobil dapat memiliki fungsi untuk berjalan, mengerem, membunyikan klakson, dll.

Class pada Javascript

Class sebetulnya adalah sebuah functions yang spesial, dan seperti function yang kita ketahui sebelumnya bahwa functions dapat dideklarasikan dan dipanggil begitu pula dengan Class.

Deklarasi Class di Javascript

Cara yang pertama untuk membuat sebuah class yaitu dengan mendeklarasikannya. Caranya adalah tuliskan "Class" diikuti dengan nama class-nya.

```
class Car {
  constructor(brand,factory) {
      this.brand = brand
      this.factory = factory
      this.sound = "honk! honk!vroomvroom"
  }
}
```

KEMNAKER

Deklarasi Class dengan variabel

Cara lain untuk membuat sebuah class yaitu dengan cara membuat sebuah variabel. Class tersebut boleh diberikan nama atau tidak diberi nama.
contohnya sebagai berikut:

```
var Car = class {
  constructor(brand, factory) {
    this.brand = brand
    this.factory = factory
  }
}
```

```
var Car = class Car2 {
  constructor(brand, factory) {
    this.brand = brand
    this.factory = factory
  }
}
```

Aturan Penamaan Class

Nama sebuah Class biasanya menggunakan kapital pada huruf pertama nya. Jika terdapat dua kata atau lebih maka huruf pertama pada kata yang selanjutnya harus kapital.

```
class Car{} // BENAR
class car{} // SALAH
class SportsCar {} // BENAR
class sportscar {} // SALAH
```

this pada class javascript

Jika diperhatikan selalu terdapat sintaks this pada contoh class di atas. this tersebut adalah sintaks untuk menunjuk class atau seluruh body di class itu sendiri. Sintaks this hanya hidup di dalam deklarasi Class dan tidak bisa dipanggil di luar class.

Di dalam deklarasi sebuah class maka cara untuk memanggil property atau methods adalah dengan menambahkan this lalu diikuti dengan titik dan nama property atau methodsnya.

constructor() pada class javascript

Constructor adalah methods yang selalu dijalankan paling pertama ketika sebuah class dipanggil (instance). Constructor dapat menerima parameter yang mengirim nilai ke dalam class tersebut.

Instance Class (membuat Object dari Class)

Seperti function pada umumnya yang terdapat deklarasi dan pemanggilan function atau instance. Deklarasi Class sudah dijelaskan pada point sebelumnya. Class juga memiliki cara untuk pemanggilannya contohnya sebagai berikut:

```
class Car {
  constructor(brand,factory) {
      this.brand = brand
      this.factory = factory
      this.sound = "honk! honk!vroomvroom"
  }
}
// Instance dari class Car
var innovam = new Car("Innovam", "Toyotwo")
console.log(innovam)
```

Method

Sintaks constructor pada class merupakan method khusus, dimana dilakukan inisialisasi properties, yang akan dieksekusi secara otomatis ketika class dibuat, dan ia harus memiliki nama "constructor". (Jika tidak dituliskan, maka Javascript akan menambahkan method constructor kosong secara otomatis).

Kita juga dapat membuat method sendiri, dengan sintaks yang sudah biasa kita gunakan

Contoh Method pada Javascript

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return "I have a " + this.carname;
  }
}

mycar = new Car("Ford");
console.log(mycar.present()) // I have a Ford
```

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present(x) {
    return x + ", I have a " + this.carname;
  }
}

mycar = new Car("Ford");
console.log(mycar.present("Hello"));
```

Getters dan Setters

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

mycar = new Car("Ford");
mycar.carname = "Volvo"; // memanggil setter, mengubah Ford menjadi Volvo
console.log(mycar.carname); // Volvo
```

Static Method

Static methods didefinisikan hanya untuk class itu sendiri.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  static hello() {
    return "Hello!!";
  }
}

mycar = new Car("Ford");

// memanggil 'hello()' pada class Car:
console.log(Car.hello());
```

Inheritance

Inheritance merupakan konsep pewarisan dimana sebuah class dapat mewarisi method maupun property dari class parentnya

Untuk membuat inheritance dari suatu class, gunakan keyword extends. Class yang dibuat dengan metode inheritance, akan memiliki method yang sama dengan class asalnya.

KEMNAKER

sanbercode

Kesimpulan

- Asynchronous
- OOP

KEMNAKER

Sanbercode

Terima Kasih